Alt/Az and Derotator Calculations

With the new generation of low-read-noise cameras, short-exposure astrophotography with long integration times for Dobsonians has become feasible. Due to the long integration times, the use of a field derotator is necessary. While rotators are fairly common for framing the image, not all rotators provide a derotator feature, and higher-level APIs in ASCOM and INDI don't have them. However, high-level clients such as Nina and Ekos often provide a scripting ability that enables the use of derotation in between short exposures. Here we will discuss the math needed to make this possible. The goal is to find the angle between lines of constant Alt (camera) and constant Dec (target) at the target based on the local time, observer location (Lat/Long), and target coordinates (RA/Dec). We will attach Scilab code that implements this.

Hour angle

The Julian date system is well described in Wikipedia. Its origin is at noon Universal Time on Jan 1, 4713 BC. The position of heavenly bodies is mostly determined by Earth's rotation and to a much smaller extent by nutation and precession. For those reasons we use the Julian-based J2000 system where the origin is reset to noon Universal Time on Jan 1, 2000. Precession and nutation since then are only a few arc minutes, which can be ignored for our purposes. We can find the Julian date by applying the J2000 system, then adding the offset of J2000 origin to the Julian origin.

Calculating today's hour angle in the J2000 system can be done by a clever formula in Wikipedia (see link above), or by simply using a table of cumulative days-per-year, one for regular years and one for leap years. The correction for the 12-hour offset of Julian relative to UT (12:00 and 0:00, respectively) is most easily done after obtaining the Julian day number since the J2000 origin. Then we add in the local time zone and daylight savings time to find UT from the local time. We refer to the attached code for the details.

Given the Julian time, we can apply the formula

$$HA = GST + LON - RA$$

where HA = local hour angle, GST = Greenwich Sidereal Time, LON = observer's longitude, and RA = Right Ascension of the target. There are two versions of GST for nutation and/or precession, but if we ignore that it is Universal Time (UT). How this relates to Earth's Rotation Angle (ERA) and the Julian date is discussed in Wikipedia. We approximate the ERA as

$$ERA = 2\pi(0.7790572732640 + 1.00273781191135448\,UT)$$

where UT is the Universal Time. Truncating this to 2π then yields the target's HA.

Transformations between Equatorial and Alt/Az systems

It is useful to think of Earth as a near-infinitely small sphere inside the unit sphere in a Cartesian coordinate system. It is convenient to represent observation targets as dots on the unit sphere because their coordinates can be readily converted to angles. Earth being very small on this scale, it resembles the night skies with targets at near-infinity. We will define two Cartesian coordinate systems labeled as XYZ and UVW.

The XYZ system is right-handed with Z pointing at the Northern Celestial Pole (NCP) and the YZ plane containing the observer's location. Thus, the Y axis points at the observer's location down South (from the observer's point of view) while the X axis points West. This is most easily related to an equatorial HA/Dec coordinate system where targets move along circles of constant Dec values.

The UVW system is also right-handed and has its W axis pointing at zenith at the observer's zenith with the UV plane tangent to Earth's surface and V pointing at the (local) South meridian. We can think of it as the XYZ turned over an angle $\frac{\pi}{2} - \lambda$ along the YZ plane where λ is the observer's latitude. The observer's V axis faces the South meridian while the U axis points West.

We choose the natural bases $\{e_X, e_Y, e_Z\}$ and $\{e_U, e_V, e_W\}$ for the XYZ and UVW systems, respectively, to relate them numerically. Here, $e_X = e_U$ and

$$(e_V, e_W) = (e_Y, e_Z)\begin{pmatrix} s & c \\ -c & s \end{pmatrix}$$

where $c = \cos\lambda$ and $s = \sin\lambda$. Thus, for a point p on the unit sphere, we have

$$p = (e_X, e_Y, e_Z)\begin{pmatrix} x \\ y \\ z \end{pmatrix} = (e_U, e_V, e_W)\begin{pmatrix} u \\ v \\ w \end{pmatrix}$$

If we define

$$T = \begin{pmatrix} 1 & 0 & 0 \\ 0 & s & c \\ 0 & -c & s \end{pmatrix}$$

then we have

$$(e_U, e_V, e_W) = (e_X, e_Y, e_Z)\, T$$

Therefore, we can relate the numerical representations of p by

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = T \begin{pmatrix} u \\ v \\ w \end{pmatrix}$$

Note that T is orthogonal, so $T^{-1} = T'$.

A representation of p in equatorial coordinates (HA, DEC) = (η, δ) is

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} \cos\delta \sin\eta \\ \cos\delta \cos\eta \\ \sin\delta \end{pmatrix}$$

And in Alt/Az coordinates is (Alt, Az) = (α, ζ) is

$$\begin{pmatrix} u \\ v \\ w \end{pmatrix} = \begin{pmatrix} \cos\alpha \sin(\zeta - \pi) \\ \cos\alpha \cos(\zeta - \pi) \\ \sin\alpha \end{pmatrix}$$

For the $-\pi$ term, note that AZ = 0 points at the North meridian while the V axis points South. The angle $\zeta - \pi$ is then somewhat similar to HA that also has its origin at the South meridian.

Going from equatorial to Alt/Az is then done by first calculating

$$\begin{pmatrix} u \\ v \\ w \end{pmatrix} = T' \begin{pmatrix} \cos\delta \sin\eta \\ \cos\delta \cos\eta \\ \sin\delta \end{pmatrix}$$

The Alt/Az coordinates are then by $\alpha = \sin^{-1} w$ and $\zeta = \tan^{-1}(u, v) + \pi$ where we use the inverse tangent for individual coordinates to cover all quadrants without ambiguity. This form is very common in numerical packages, see for instance Scilab. Note that the angle $\tan^{-1}(u, v)$ is with the V axis in the U (West) direction, so we have to add π to get Az.

Going from Alt/Az to equatorial coordinates we first calculate

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = T \begin{pmatrix} \cos\alpha \sin(\zeta - \pi) \\ \cos\alpha \cos(\zeta - \pi) \\ \sin\alpha \end{pmatrix}$$

then retrieve $\delta = \sin^{-1} z$ and $\eta = \tan^{-1}(x, y)$.

Rotator angles

Since the rotator angles depend on user preferences for the specific target, we use the angle between the Alt/Az grid and the RA/Dec grid at the target as a reference. The user can then add his additional rotation preference for framing to that angle. We assume short exposures where the coordinates are essentially constant during the exposure. The Alt/Az grid represents the position of the camera, whereas the RA/Dec grid represents the orbit of the target. Note that in HA, the target is moving in the opposite direction of increasing RA because the HA = 0 line moves towards increasing RA in function of time. This is important when validating the results with planetary programs such as Stellarium.

We will use the Alt/Az axes as a 2D Cartesian coordinate system. The X axis will be Az (positive direction is that of increasing Az), the Y axis will be Alt (positive direction is that of increasing Alt). Note that they are perpendicular. The direction in which the camera moves is that of decreasing RA, which is the same as increasing HA as time moves along. We can use the unit vectors of the XY axes to form inner products with the HA line and retrieve the angle from the positive Az axis to the positive HA (negative RA) axis for all quadrants.

The tangent to the Az curve (constant Alt) follows from its derivative to ζ:

$$\frac{dp}{d\zeta} = \begin{pmatrix} \cos\alpha\cos(\zeta-\pi) \\ -\cos\alpha\sin(\zeta-\pi) \\ 0 \end{pmatrix}$$

The tangent to the Alt curve (constant Az) follows from its derivative to α:

$$\frac{dp}{d\alpha} = \begin{pmatrix} -\sin\alpha\sin(\zeta-\pi) \\ -\sin\alpha\cos(\zeta-\pi) \\ \cos\alpha \end{pmatrix} = \begin{pmatrix} \sin\alpha\sin(\zeta) \\ \sin\alpha\cos(\zeta) \\ \cos\alpha \end{pmatrix}$$

The tangent to the HA curve (constant Dec) follows from its derivative to η:

$$\frac{dp}{d\eta} = T' \begin{pmatrix} \cos\delta\cos\eta \\ -\cos\delta\sin\eta \\ 0 \end{pmatrix}$$

Note that this is done in the UVW coordinate system, which is why the last term was pre-multiplied by T' and the earlier ones were not. If we normalize the vectors $\frac{dp}{d\zeta}$, $\frac{dp}{d\alpha}$ and $\frac{dp}{d\eta}$ to vectors e_ζ, e_α, and e_η, respectively, then the angle φ from e_ζ to e_η is

$$\varphi = \tan^{-1}\left(e'_\eta e_\alpha,\ e'_\eta e_\zeta\right)$$

Again, when comparing this with Stellarium keep in mind that η runs opposite to increasing RA so the direction in which the target moves relative to the constant Alt line is at an angle $-\varphi$. For the derotation the direction does not really matter so long as we we adjust the framing offset that the user wants accordingly.

Scilab code

File: altAz.sce

```
mode(0);
exec "coordinates.sce";

// Santa Barbara latitude/longitude according to Stellarium
lat = dms2rad(34, 25, 14.99);      // Santa Barbara latitude in radians
lon = dms2rad(-119, -41, -53.48);  // Santa Barbara longitude in radians
```

```
// Local time in Santa Barbara (set as fixed in Stellarium)
tz = -8;
ds = 0;
y = 2026;
m = 2;
d = 13;
h = 21;
mi = 0;
s = 0;

// Capella J2000
name = "Capella";
ra_hms  = [05, 16, 42.4];
dec_dms = [45, 59, 49.5];
// Dubhe J2000
name = "Dubhe";
ra_hms  = [11, 03, 45.71];
dec_dms = [61, 44, 52.7];
// Rigel J2000
name = "Rigel";
ra_hms  = [05, 14, 32.86];
dec_dms = [-8, -12, -15.0];

// Convert to radians
ra  = hms2rad( ra_hms(1),  ra_hms(2),  ra_hms(3));
dec = dms2rad(dec_dms(1), dec_dms(2), dec_dms(3));

[ha, jd, jd2000, njd2000] = ra2ha(ra, lon, tz, ds, y, m, d, h, mi, s);
ha_hms = rad2hms(ha);

// Convert ha/dec to Alt/Az
[alt, az, xyz1] = eq2altaz(ha, dec, lat);
alt_dms = rad2dms(alt);
az_dms  = rad2dms(az, %t);
// Convert alt/az back to ha/dec
[ha1, dec1, xyz2] = altaz2eq(alt, az, lat);
ha1_hms = rad2hms(ha1);
dec1_dms = rad2dms(dec1);

// Note:  We implemented the signs to be compatible with Stellarium
printf("Target: %s\n", name);
printf("RA  = %g hrs, %g min, %g sec\n",  ra_hms(1),   ra_hms(2),   ra_hms(3));
printf("DEC = %g deg, %g min, %g sec\n", dec_dms(1),  dec_dms(2),  dec_dms(3));
printf("HA  = %g hrs, %g min, %g sec\n",  ha_hms(1),   ha_hms(2),   ha_hms(3));
printf("ALT = %g deg, %g min, %g sec\n", alt_dms(1),  alt_dms(2),  alt_dms(3));
printf("AZ  = %g deg, %g min, %g sec\n",  az_dms(1),   az_dms(2),   az_dms(3));
printf("DEC1= %g deg, %g min, %g sec\n", dec1_dms(1), dec1_dms(2), dec1_dms(3));
printf("HA1 = %g hrs, %g min, %g sec\n",  ha1_hms(1),  ha1_hms(2),  ha1_hms(3));

[phi] = rotatorAngle(lat, ha, dec, alt, az);
printf("Angle of HA+ line in (Az+, Alt+) frame = %g degrees\n", 180/%pi*phi);

return

// The code below is an example of how the Alt/Az coordinates change
// for targets of various declinations.

// Alt/Az speed profiles for stars on the -45 degree hour angle
az = zeros(91, 91);
alt = zeros(91, 91);
for di = [0:90] do
```

```
  for hai = [-45:45] do
    d = di*%pi/180;
    ha = hai*%pi/180;
    [alti, azi] = eq2altaz(ha, d, lat);
    i = hai+45+1;
    j = di+1;
    if (i > 1) then
      if (azi > az(i-1,j) + %pi) then
        azi = azi - 2*%pi;
      end
      if (azi < az(i-1,j) - %pi) then
        azi = azi + 2*%pi;
      end
    end
    alt(i, j) = alti;
    az(i, j) = azi;
  end
end

fs = 4;

clf(0); scf(0); plot([-45:45]', alt*180/%pi);
a = gca();
a.y_label.text = "Altitude (deg)";
a.x_label.text = "Hour angle (deg)";
a.title.text = "Altitude vs. Hour Angle";
a.y_label.font_size = fs;
a.x_label.font_size = fs;
a.title.font_size = fs;

clf(1); scf(1); plot([-45:45]', az*180/%pi);
a = gca();
a.y_label.text = "Azimuth (deg)";
a.x_label.text = "Hour angle (deg)";
a.title.text = "Azimuth vs. Hour Angle";
a.y_label.font_size = fs;
a.x_label.font_size = fs;
a.title.font_size = fs;

clf(2); scf(2); plot([-44:45]'-0.5, 0.25*(alt(2:91,:)-alt(1:90,:))*180/%pi);
a = gca();
a.y_label.text = "Alt rate (deg/min)";
a.x_label.text = "Hour angle (deg)";
a.title.text = "Altitude rate vs. Hour Angle";
a.y_label.font_size = fs;
a.x_label.font_size = fs;
a.title.font_size = fs;

clf(3); scf(3); plot([-44:45]'-0.5, 0.25*(az(2:91,:)-az(1:90,:))*180/%pi);
a = gca();
a.y_label.text = "Az rate (deg/min)";
a.x_label.text = "Hour angle (deg)";
a.title.text = "Azimuth rate vs. Hour Angle";
a.y_label.font_size = fs;
a.x_label.font_size = fs;
a.title.font_size = fs;

// Z12 Alt/Az speed analysis
D1 = 630;             // Az board diameter in mm
r1 = D1/2;            // Radius
dpm = 4;              // Degrees per minute (see speed profile)
```

```
l1 = 2*%pi*r1*(dpm/360); // Travel in mm/min at the base wheel circumference
rpm = 2;              // Motor external axis RPM
r2 = l1/(2*%pi*rpm);    // Radius required to achieve l1 at 2 rpm
D2 = 2*r2;            // Diameter required
```

File: coordinates.sce

```
////////////////////////////////////////////////////////////////
// Coordinate transformation between equatorial and Alt/Az.
////////////////////////////////////////////////////////////////
// ha      = hour angle (0 at South meridian, positive towards West)
// dec     = declination (-90 at SCP to +90 at NCP)
// lambda  = latitude (of the observer)
// az      = Azimuth (0 at North meridian, positive towards the East)
// alt     = Altitude (0 at horizon, positive towards zenith)
// Positive x = towards the West
// Positive y = towards the South
// Positive z = towards the zenith
//
// We start with a right-handed Cartesian XYZ coordinate system where
// Z is pointing to the NCP and where the observer at latitude lambda
// is in the YZ plane.  Let the basis of the coordinate system be
// the unit vectors in each direction {ex, ey, ez}.
//
// Next, we define a right-handed Cartesian UVW coordinate system
// with corresponding natural base vectors {eu, ev, ew} where
// eu = ex and ev and ew are ey and ez rotated around the X axis
// over an angle pi/2 - lambda such that ew points at zenith at
// latitude lambda.  Thus, if we define s = sin(lambda) and
// c = cos(lambda), then ev = s*ey - c*ez and ew = c*ey + s*ez.
// In matrix notation, [ev,ew] = [ey,ez]*[s,c;-c,s].
//
// A vector p has representations in both coordinate systems
// p = [ex ey ez]*[x;y;z] = [eu ev ew]*[u;v;w].  If we define
// T = [1,0,0; 0,s,c; 0,-c,s] then [eu, ev, ew] = [ex, ey, ez]*T
// therefore [x;y;z] = T*[u;v;w].  Note that T is orthogonal, so
// inv(T) = T'.
//
// Let us define polar coordinates in hour angle (ha) and
// declination (dec) such that in the XYZ coordinates we have
// [x;y;z] = [cos(dec)*sin(ha); cos(dec)*cos(ha); sin(dec)].
// Let us also define polar coordinates in Azimuth (az) and
// Altitude (alt) such that in the UVW coordinates we have
// [u;v;w] = [cos(alt)*sin(az-%pi); cos(alt)*cos(az-%pi); sin(alt)].
// The minus sign for u and v is because az = 0 points North
// whereas ha = 0 points South.
//
// These relationships let us convert ha/dec to alt/az and vice versa.
// Summarized,
// c = cos(lambda), s = sin(lambda), T = [1,0,0; 0,s,c; 0,-c,s]
// [x;y;z] = T*[u;v;w]
// [x;y;z] = [cos(dec)*sin(ha); cos(dec)*cos(ha); sin(dec)]
// [u;v;w] = [-cos(alt)*sin(az); -cos(alt)*cos(az); sin(alt)]

// Radians to degrees, minutes, seconds (0-360 degrees)
// This is usually used for DEC where we can have negative
// signs, but not for Az calculations that are positive
function dms=rad2dms(r, positive)
  if ~exists('positive', 'l') then
```

```
      positive = %f;
   end
   r = modulo(r, 2*%pi);
   if positive & (r < 0) then
      r = r + 2*%pi;
   end
   deg = r*180/%pi;   // Radians to degrees
   d = int(deg);      // Truncate degrees
   mf = deg - d;      // Minutes fraction
   m = int(60*mf);    // Truncate minutes
   sf = mf - m/60;    // Seconds fraction
   s = 3600*sf;       // Seconds
   dms = [d, m, s];
endfunction

// Radians to hours, minutes, seconds (0-24 hours)
// Note: 1 hour = 15 degrees = 60 minutes(4 minutes per degree)
//       as opposed to the usual 60 minutes per degree
function hms=rad2hms(r)
   r = modulo(r, 2*%pi);
   if r < 0 then
      r = r + 2*%pi;
   end
   deg = r*180/%pi;   // Radians to degrees
   h = int(deg/15);   // Truncate hours
   hf = (deg - 15*h)/15;  // Hour fraction
   m = int(60*hf);    // Truncate minutes
   sf = hf - m/60;    // Seconds fraction
   s = 3600*sf;       // Seconds
   hms = [h, m, s];
endfunction

// Degrees, minutes, seconds to radians
function r=dms2rad(d, m, s)
   r = (%pi/180)*(d + m/60 + s/3600);
endfunction

// Hours, minutes, seconds to radians
// Note: 1 hour = 15 degrees = 60 minutes(4 minutes per degree)
//       as opposed to the usual 60 minutes per degree
function r=hms2rad(h, m, s)
   r = (%pi/180)*(h + m/60 + s/3600)*15;
endfunction

// Equatorial to Alt/Az
function [alt, az, xyz]=eq2altaz(ha, dec, lambda)
   // c = cos(lambda), s = sin(lambda), T = [1,0,0; 0,s,c; 0,-c,s]
   // [x;y;z] = T*[u;v;w]
   // [x;y;z] = [cos(dec)*sin(ha); cos(dec)*cos(ha); sin(dec)]
   // [u;v;w] = [cos(alt)*sin(az-%pi); cos(alt)*cos(az-%pi); sin(alt)]
   c = cos(lambda);
   s = sin(lambda);
   T = [1,0,0; 0,s,c; 0,-c,s];
   xyz = [cos(dec)*sin(ha); cos(dec)*cos(ha); sin(dec)];
   uvw = T'*xyz;
   u = uvw(1); v = uvw(2); w = uvw(3);
   alt = asin(w);
   az = atan(u, v) + %pi;
endfunction

// Alt/Az to equatorial
```

```
function [ha, dec, xyz]=altaz2eq(alt, az, lambda)
   // c = cos(lambda), s = sin(lambda), T = [1,0,0; 0,s,c; 0,-c,s]
   // [x;y;z] = T*[u;v;w]
   // [x;y;z] = [cos(dec)*sin(ha); cos(dec)*cos(ha); sin(dec)]
   // [u;v;w] = [cos(alt)*sin(az-%pi); cos(alt)*cos(az-%pi); sin(alt)]
   c = cos(lambda);
   s = sin(lambda);
   T = [1,0,0; 0,s,c; 0,-c,s];
   uvw = [cos(alt)*sin(az-%pi); cos(alt)*cos(az-%pi); sin(alt)];
   xyz = T*uvw;
   x = xyz(1); y = xyz(2); z = xyz(3);
   dec = asin(z);
   ha = atan(x, y);
endfunction

////////////////////////////////////////////////////////////////
// RA to hour angle
////////////////////////////////////////////////////////////////
function n=days_since_2000(y, m, d)
   n1 = int((y+3-2000)/4);    // Leap years since 2000
   n2 = (y - 2000 - n1);      // Non-leap years since 2000
   // Cumulative days up to a give month (as array index)
   if (modulo(y, 4) == 0) then
      md = [0 31 60 91 121 152 182 213 244 274 305 335];
   else
      md = [0 31 59 90 120 151 181 212 243 273 304 334];
   end
   n = n1*366 + n2*365 + d + md(m) - 1;
endfunction

function [jd, njd]=julianDate2000(y, m, d, tz, ds, y, m, d, h, mi, s)
   // The time parameters are the local time.  The correction for
   // local time to UTC time is most easily done in the Julian date,
   // so we first calculate the Julian date pretending as if it's the
   // UTC time and later apply the correction to local time.
   // https://en.wikipedia.org/wiki/Julian_day#Julian_day_number_calculation
   //
   // Cumulative days up to a given month, indexed by month
   if (modulo(y, 4) == 0) then
      md = [0 31 60 91 121 152 182 213 244 274 305 335];
   else
      md = [0 31 59 90 120 151 181 212 243 273 304 334];
   end
   // Pretend as if the time parameters represent UTC time
   // Days since 2000 Jan 1 0:00
   n1 = int((y+3-2000)/4);    // Leap years since 2000
   n2 = (y - 2000 - n1);      // Non-leap years since 2000
   njd = n1*366 + n2*365 + d + md(m) - 1;
   // Correct njd for the difference between local and UTC time
   h = h - tz - ds;
   if h > 23 then
      njd = njd + 1;
      h = h - 24;
   end
   if h < 0 then
      njd = njd - 1;
      h = h + 24;
   end
   // Convert UTC time (base = 0:00) to Julian time (base = 12:00)
   h = h - 12;
   if h < 0 then
```

```
    njd = njd - 1;
    h = h + 24;
  end
  jd = njd + (h + mi/60 + s/3600)/24;
endfunction

function [jd, jdn]=julianDate1(y, m, d, tz, ds, y, m, d, h, mi, s)
  // Crazy formula for Julian date from UTC time - not used yet
  // https://en.wikipedia.org/wiki/Hour_angle#Relation_with_right_ascension
  tmp0 = int((m-14)/12);
  tmp1 = int((1461*y + 4800 + tmp0)/4);
  tmp2 = int((367*(m-2-12*tmp0))/12);
  tmp3 = int(3*int((y+4900+tmp0)/100)/4);
  jdn = tmp1 + tmp2 - tmp3 + d - 32075;
  jd = jdn + (h-12)/24 + mi/1440 + s/86400;
endfunction

function [ha, jd, jd2000, njd2000]=ra2ha(ra, lon, tz, ds, y, m, d, h, mi, s)
  // 1) Equinox is when day and night are equally long, when the plane
  //   of the Earth's equator passes through the center of the sun.
  //   Equinoctal point: location of the sun during an equinox.
  //   Vernal equinoctal point: RA=0, longitude=0.
  //   Autumnal equinoctal point: RA=12h, longitude=180.
  // 2) A siderial day is the time it takes Earth to make one rotation
  //   relative to the vernal equinox: 23 hours, 56 minutes, 4.0916
  //   seconds (23.9344699 hours or 0.99726958 mean solar days).
  //   Apparent siderial time = hour angle of the vernal equinox.
  //   The mean sidereal time accounts for the Earth's nutation.
  //   Apparent and mean sidereal time differ by at most 1.2 seconds.
  //   GMST = Greenwich mean sidereal time.
  // 3) The Julian Day Number (JDN) is the integer assigned to a whole
  //   solar day in the Julian day count starting from noon Greenwich
  //   Mean Time, with Julian day number 0 assigned to the day starting
  //   at noon on January 1, 4713 BC, proleptic Julian calendar
  //   (November 24, 4714 BC, in the proleptic Gregorian calendar).
  //   For example, the Julian day number for January 1, 2000,
  //   was 2451545.
  // 4) UT1 is proportional to the rotational angle of the Earth
  //   relative to the ICRF (International Celestial Reference Frame).
  //   ERA = Earth's Rotational Angle.
  //   ERA = 2*pi*(0.7790572732640 + 1.00273781191135448*TU) radians
  //   where TU =  (Julian UT1 date - 2451545.0).
  //   GMST = 18.697374558 + 24.06570982441908 * D where D is the
  //   interval, in UT1 days including any fraction of a day, since
  //   2000 January 1, at 12h UT.

  // Julian date since 2000 Jan 1, noon UTC
  // https://en.wikipedia.org/wiki/Julian_day#Julian_day_number_calculation
  [jd2000, njd2000] = julianDate2000(y, m, d, tz, ds, y, m, d, h, mi, s);
  jd = jd2000 + 2451545.0;

  // Earth rotation angle
  // https://en.wikipedia.org/wiki/Sidereal_time#Earth_rotation_angle
  era = 2*%pi*(0.7790572732640 + 1.00273781191135448*jd2000);

  // Hour angle - we simply take the era and ignore precession / nutation
  // https://en.wikipedia.org/wiki/Hour_angle#Relation_with_right_ascension
  ha = era + lon - ra;
  ha = modulo(ha, 2*%pi);
  if ha < 0 then
    ha = ha + 2*%pi;
```

```
  end
endfunction

function [phi]=rotatorAngle(lat, ha, dec, alt, az)
  c = cos(lat);
  s = sin(lat);
  T = [1,0,0; 0,s,c; 0,-c,s];    // [x;y;z] = T*[u;v;w]

  // Find the normalized tangent to the target's orbit in UVW coordinates
  xyz = [cos(dec)*sin(ha); cos(dec)*cos(ha); sin(dec)];
  uvw = T'*xyz;
  d_ha = T'*[cos(dec)*cos(ha); -cos(dec)*sin(ha); 0];
  d_ha = d_ha/norm(d_ha);

  // Find the Az motion of the camera sensor in UVW coordinates
  uvw = [cos(alt)*sin(az-%pi); cos(alt)*cos(az-%pi); sin(alt)];
  d_az = [cos(alt)*cos(az-%pi); -cos(alt)*sin(az-%pi); 0];
  d_az = d_az/norm(d_az);
  d_alt = [sin(alt)*sin(az); sin(alt)*cos(az); cos(alt)];

  // Angle between d_ha and d_az
  d_ha_az = d_ha'*d_az;
  d_ha_alt = d_ha'*d_alt;
  phi = atan(d_ha_alt, d_ha_az);
  // Note: HA increases in the opposite direction to RA because the
  //       ha=0 meridian (local South) moves towards the East in RA.
endfunction
```